Design and Implementation Tradeoffs for Wide-Area Resource Discovery

David Oppenheimer[†], Jeannie Albrecht[‡], David Patterson[†], and Amin Vahdat[‡]
[†]EECS Computer Science Division
University of California Berkeley
{davidopp,patterson}@cs.berkeley.edu
[‡]Department of Computer Science and Engineering
University of California San Diego
{jalbrecht,vahdat}@cs.ucsd.edu

Abstract

This paper describes the design and implementation of SWORD, a scalable resource discovery service for wide-area distributed systems. In contrast to previous systems, SWORD allows users to describe desired resources as a topology of interconnected groups with required intra-group, inter-group, and per-node characteristics, along with the utility that the application derives from various ranges of values of those characteristics. This design gives users the flexibility to find geographically distributed resources for applications that are sensitive to both node and network characteristics, and allows the system to rank acceptable configurations based on their quality for that application. We explore a variety of architectures to deliver SWORD's functionality in a scalable and highly-available manner. A 1000-node ModelNet evaluation using a workload of measurements collected from PlanetLab shows that an architecture based on 4-node server cluster sites at network peering facilities outperforms a decentralized DHT-based resource discovery infrastructure for all but the smallest number of sites. While such a centralized architecture shows significant promise, we find that our decentralized implementation, both in emulation and running continuously on over 200 PlanetLab nodes, performs well while benefitting from the DHT's self-healing properties.

1. Introduction

Grid applications, ranging from distributed scientific computations [7, 14, 26, 32] to long-running network services [16, 20, 22, 25, 34], are becoming an increasingly important part of the computing landscape. At the same time, the prevalence, scale, and geographic distribution of deployment platforms for such applications has grown. For example, Grid3 consists of 2000 CPUs at 25 sites [18], and PlanetLab offers 500 machines at 200 sites [3]. One significant difficulty in the practical use of such shared, large-scale infrastructures centers around *resource discovery* and *service placement*, that is, locating an appropriate subset of the platform to host a distributed computation, experiment, or service. We refer to this combined task of resource discovery and service placement as simply resource discovery.

This paper considers the architectural tradeoffs involved in building a resource discovery system targeting Grid settings. Such a system must *scale* to thousands of nodes and sites. It must be *highly available*, as it is the system entry point for launching and maintaining applications. The system must *accurately report* dynamically changing node and network characteristics. Because many distributed applications are sensitive to inter-node latency and bandwidth, the system must support queries over not just per-node characteristics, but also over *inter-node characteristics*. Finally, because applications have widely varying needs and because desired resources are often scarce, users should be able to specify the ranges of resource quantities that their application needs, as well as the *utility* lost from the selection of imperfect but acceptable nodes.

A number of recent efforts have explored global resource discovery [2, 9, 19, 23, 41, 46, 47]. However, to our knowledge no existing system meets all of the above requirements. In particular, resource discovery systems must support specifying required inter-node characteristics and the relative utility of both per-node and inter-node characteristics in assessing tradeoffs among competing potential configurations. One contribution of this work is to present the semantics of SWORD, a resource discovery infrastructure that allows users to easily describe desired resources as a *topology of interconnected groups* with required intra-group and inter-group characteristics and *penalty functions* to indicate the utility of those characteristics to the application. Our implementation of SWORD, which supports our own query language and Condor ClassAds, has been deployed as a continuously-running PlanetLab service for more than six months, tracking over 40 metrics per machine from a combination of sources [11, 33, 40, 42].

A resource discovery service could incur significant load, both in terms of monitoring the target infrastructure and answering resource discovery queries. Thus, the

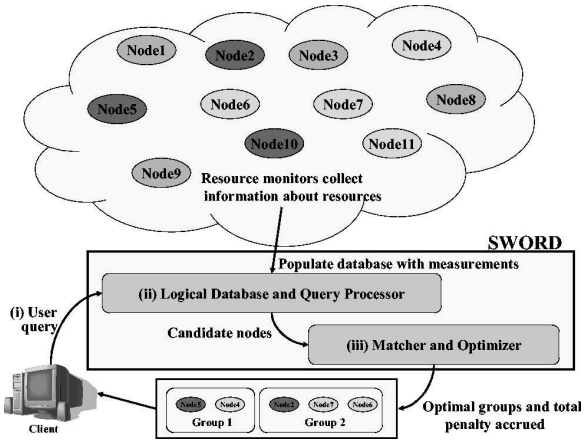

Figure 1. High-level SWORD architecture.

second principal contribution of this work is an exploration of the architectural space for efficient and robust resource discovery. Our initial intuition was that some variety of a distributed architecture would be required to scale to our target settings, an approach adopted by related efforts [2, 19, 41, 46]). The appropriate architecture depends upon the system workload and deployment scenario, and distributed architectures do offer a number of advantages. However, through a large-scale deployment and evaluation of multiple instances of SWORD embodying different architectures, we find that a centralized architecture performs competitively with several decentralized architectures under a number of realistic settings.

The rest of this paper is organized as follows. Section 2 presents an overview of the SWORD architecture, and Section 3 describes our implementation. Section 4 evaluates SWORD, Section 5 presents related work, and Section 6 concludes.

2. System Architecture and Queries

2.1. Target Usage Scenarios

SWORD is designed with two usage scenarios in mind. When used in a "best effort" environment such as PlanetLab, SWORD matches a user's resource specification to the node characteristics at the time the request is made. SWORD returns an ordered list consisting of sets of nodes. The list is ranked by the closeness of each set's match to user desires. Users deploy their application on the set of nodes deemed to best match their requirements and periodically re-query SWORD to realign their configuration with time-varying system characteristics.

Alternatively, SWORD may be used in conjunction with an external resource allocation or admission control mechanism. The resource allocation system might augment SWORD's measurement database with information about which resources each user can access, at what cost, and during what time periods. Queries specify a period of time for using the requested resources, and candidate nodes are filtered to exclude nodes that are not available to that user for the requested period. Although our deployment of SWORD has been in a "best effort" environment, we expect to integrate it with resource allocation tools such as SHARP [17] or SNAP [10] to support arbitrated usage scenarios.

2.2. Query Semantics and Architecture

Two observations about distributed scientific applications and distributed network services guide the design of SWORD's query representation. First, as also noted in [23], some distributed scientific applications desire collections of tightly-coupled groups of nodes. To enable discovery of such groups, SWORD supports requests for a number of equivalence classes called *groups*, each defined by a required number of nodes, a range of acceptable per-node characteristics, and acceptable inter-node measurements such as latency and bandwidth among group nodes. Some applications may further wish to constrain group location in the network topology or connectivity *among* groups. For example, a Content Distribution Network such as CoDeeN [34] may desire small clusters of tightly-coupled nodes near several distinct geographically distributed user populations, with low-latency, high-bandwidth links among clusters. Hence, SWORD also allows the user to request that each group be within a desired latency of a specified network reference point, and to specify required ranges of inter-node measurements between nodes in distinct groups.

A second observation guiding SWORD's query language is varying application sensitivity to deviations from specified per-node and inter-node constraints. Thus, we wish to allow users to specify absolute requirements on per-node and inter-node characteristics, similar to ClassAd [36] "constraints," and sensitivity to deviations from preferred per-node and inter-node characteristics, similar to ClassAd ranking functions. In this way, nodes whose value for an attribute fall within the required range are ranked by their suitability based on their deviation from the preferred range. Users describe this sensitivity using per-attribute *penalty functions*, which can be thought of as the inverse of utility functions. Penalty functions are described in detail in [31].

SWORD consists of three parts, illustrated schematically in Figure 1. A user describes their desired configuration using a native SWORD XML syntax or a Condor ClassAd, and the system returns the lowest-penalty mapping (or a ranked set of mappings) of available nodes to groups in the user's query.

```
Group NA
   NumMachines 4
   Required Load [0.0, 2.0]
   Preferred Load [0.0, 1.0], penalty 100.0
   Required FreeDisk [500.0, MAX]  (MB)
   Preferred FreeDisk [1000.0, MAX], penalty 100.0
   Required OS [''Linux'']
   Required AllPairs Latency [0.0, 20.0] (ms)
   Preffered AllPairs Latency [0.0, 10.0], penalty 100.0
   Required AllPairs BW [0.5, MAX] (Mb/s)
   Preffered AllPairs BW [1.0, MAX], penalty 2.0
   Required Location [''NorthAmerica'', 0.0, 50.0] (ms)

Group Europe
   NumMachines 4
   Required Load [0.0, 2.0]
   Preferred Load [0.0, 1.0], penalty 100.0
   Required FreeDisk [300.0, MAX]  (MB)
   Preferred FreeDisk [1000.0, MAX], penalty 100.0
   Required OS [''Linux'']
   Required AllPairs Latency [0.0, 20.0] (ms)
   Preffered AllPairs Latency [0.0, 10.0], penalty 100.0
   Required AllPairs BW [0.5, MAX] (Mb/s)
   Preffered AllPairs BW [1.0, MAX], penalty 2.0
   Required Location [''Europe'', 0.0, 50.0]

InterGroup
    Required OnePair BW NA Europe [3.0, MAX] (Mb/s)
    Preferred OnePair BW NA Europe [5.0, MAX], penalty 0.5
```

(a)

```
<QUERY> ::= <GRP>+ <INTERGRP>?
<GRP> ::= <GRPNAME> <NUMMACHINES>? (<REQ> <PREF>?)+
<GRPNAME> ::= 'Group' <GNAME>
<GNAME> ::= <STRING>
<NUMMACHINES> ::= 'NumMachines' <INTEGER>
<REQ> ::= <PERREQ> | <INTERREQ>
<PERREQ> ::= 'Required' <ATTRNAME> [<DOUBLE> | 'Min', <DOUBLE> | 'Max'] |
             'Required' <ATTRNAME> ['True'? 'False'?] |
             'Required' <ATTRNAME> [<STRING>+] |
             'Required' <ATTRNAME> [<REFPOINT>+]
<REFPOINT> ::= <LOCATION>, <DOUBLE> | 'Min', <DOUBLE> | 'Max'
<INTERREQ> ::= 'Required' 'OnePair' <ATTRNAME> [<DOUBLE> | 'Min', <DOUBLE> | 'Max'] |
               'Required' 'AllPairs' <ATTRNAME> [<DOUBLE> | 'Min', <DOUBLE> | 'Max']
<PREF> ::= <PERPREF> | <INTERPREF>
<PERPREF> ::= 'Preferred' <ATTRNAME> [<DOUBLE> | 'Min', <DOUBLE> | 'Max'],
                  'penalty' <PENALTY> |
              'Preferred' <ATTRNAME> [('True', <PENALTY>)? ('False', <PENALTY>)?] |
              'Preferred' <ATTRNAME> [(<STRING>+, <PENALTY>)+] |
              'Preferred' <ATTRNAME> [(<REFPOINT>+, <PENALTY>)+]
<INTERPREF> ::= 'Preferred' 'OnePair' <ATTRNAME> [<DOUBLE> |
                    'Min', <DOUBLE> | 'Max'], <PENALTY> |
                'Preferred' 'AllPairs' <ATTRNAME>
                    [<DOUBLE> | 'Min', <DOUBLE> | 'Max'], 'penalty' <PENALTY>
<INTERGRP> ::= (<IREQ> <IPREF>*)+
<IREQ> ::= 'Required' 'OnePair' <ATTRNAME> <GNAME> <GNAME>
                [<DOUBLE> | 'Min', <DOUBLE> | 'Max'] |
           'Required' 'AllPairs' <ATTRNAME>  <GNAME> <GNAME>
                [<DOUBLE> | 'Min', <DOUBLE> | 'Max']
<IPREF> ::= 'Preferred' 'OnePair' <ATTRNAME> <GNAME> <GNAME>
              [<DOUBLE> | 'Min', <DOUBLE> | 'Max'], 'penalty' <PENALTY> |
            'Preferred' 'AllPairs' <ATTRNAME> <GNAME> <GNAME>
              [<DOUBLE> | 'Min', <DOUBLE> | 'Max'], 'penalty' <PENALTY>
<PENALTY> ::= <DOUBLE>
```

(b)

Figure 2. (a) Sample SWORD query requests two groups. Group NA requests four Linux nodes within 50 ms (using network coordinates) of a reference node in North America. The query shows a preferred and required range of values for desired attributes, and a penalty for being outside of the preferred range. For example, nodes should have load less than 1.0, but the operator is willing to accept nodes with load up to 2.0, with a normalized penalty of 100 units for load values between 1.0 and 2.0. Group Europe is similarly defined. One inter-group bandwidth constraint between NA and Europe is specified. (b) An abstract specification of SWORD's native query language in EBNF.

2.3. Expressing Queries

Putting the group and penalty function features together, consider the following example. A small Internet search engine has large user populations in North America and Europe. The service architecture is such that groups of four nodes constitute a "site," sharing the search index and parallelizing the search functionality. The service operator decides to request one group of four nodes near a network reference point in North America, and another group near a point in Europe. To enable close cooperation within a site, the operator specifies low-latency, high-bandwidth links among all nodes within a site. Because sites periodically communicate newly-crawled data, the operator requires at least one high-bandwidth link connecting a pair of nodes in different groups. Finally, the operator places per-node requirements on free disk space and load.

SWORD supports two query languages. Figure 2(a) shows a sample query issued by the search engine operator using SWORD's XML specification language, slightly modified for clarity. Figure 2(b) provides an EBNF description of this query syntax. SWORD endeavors to locate the lowest-penalty mapping of available nodes to groups in the user's query, with the overall penalty of a mapping defined as the sum over all groups of each member node's penalty, accounting for per-node, inter-node, and inter-group constraints. For compatibility with existing Grid applications, SWORD also supports Condor ClassAds as depicted in Figure 3 for the search engine query. Because the standard ClassAd syntax does not support specifying inter-node properties, SWORD can only constrain and rank based on per-node properties using this syntax.

3. Implementation

The main components of SWORD are the *query processor* and the *optimizer*. The query processor finds all nodes matching the user's specifications. The optimizer finds the lowest penalty mapping of those candidate nodes to groups, based on specified penalty functions.

3.1. Query Processor

A node that wishes to offer its resources through SWORD joins the SWORD infrastructure and collects resource-monitoring data locally. This *reporting node* periodically sends a *measurement report* to one or more SWORD *servers*. A node need not be part of the SWORD infrastructure to submit queries. To issue a query, a *client* node sends a query to any node in the SWORD infrastructure. This *query node* receiving the request acts as a proxy into the SWORD infrastructure, potentially issuing

```
{ [ name = ''NA'';
    nummachines = 4;
    constraint = load <= 2.0 && freedisk >= 500 && os == ''linux'' &&
       netdist(''NorthAmerica'') < 50;
    rank = (100.0 * (load-1 > 0 ? load-1 : 0)) +
           (100.0 * (1000-freedisk > 0 ? 1000-freedisk : 0));
  ];

  [ name = ''Europe'';
    nummachines = 4;
    constraint = load <= 2.0 && freedisk >= 300 && os == ''linux'' &&
       netdist(''Europe'') < 50;
    rank = (100.0 * (load-1 > 0 ? load-1 : 0)) +
           (100.0 * (1000-freedisk > 0 ? 1000-freedisk : 0));
  ];
};
```

Figure 3. Sample query using ClassAd syntax.

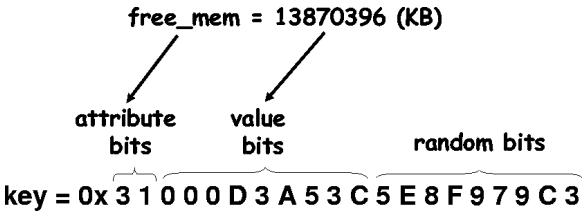

Figure 4. Constructing a DHT key from an <attribute,value> pair. This key is 80 bits for clarity; SWORD uses an analogous mapping technique to produce its 160-bit keys.

sub-queries to remote servers, depending on the query processor algorithm in use. In our deployments, the reporting node, server, client, and query node roles are implemented on the same set of nodes.

3.1.1. Design Alternatives for Storing and Retrieving Per-Node Attributes

The core query processing primitive of interest is *multi-attribute range search*. For example, consider a query that requests a group of nodes that have load average between 0.0 and 2.0 and free disk space between 100 and 200 MB.

One approach to handling such a query is a centralized architecture in which a central database collects and stores all reports of load and free disk space (along with the identity of the sending node and a timestamp), and maintains indices on load and disk space to quickly find nodes in required ranges. Although simple to implement, a centralized architecture potentially suffers from limited scalability and availability.

To improve scalability and availability, one might distribute the database across nodes. In this case, we partition the data space among servers by attribute, and then sub-partition by value. For example, one server might be responsible for handling updates for all machines whose load is between 0.0 and 1.0, another for loads between 1.0 and 2.0, etc.; another for free disk between 0 MB and 100 MB, another for free disk between 100MB and 200MB, etc. Although more scalable and available, satisfying distributed range queries in this architecture could require contacting many servers, increasing the amount of communication and the complexity of the searching algorithm.

One way to reduce implementation complexity for this approach is to use a distributed hash table (DHT) to automatically partition the data space among servers. DHTs are scalable, self-configuring, and highly available, and these traits are a good match to our target of large federated platforms that eschew centralized management. SWORD maps an <attribute,value> measurement to a server by first mapping it to a DHT key, as described below, and then asking the DHT to route the measurement to the corresponding key. The DHT delivers the message to the node responsible for the key, and automatically repartitions the keyspace among nodes when DHT nodes fail, recover, or join. To enable a query to visit the nodes responsible for a contiguous range of keys, SWORD uses existing DHT "successor pointers" that organize the nodes into a linked list sorted by ascending key ranges.[1]

Figure 4 shows how SWORD maps a measurement to a DHT key. The top bits partition attributes evenly among DHT servers. The middle bits map a value to one of the servers responsible for that attribute. The bottom bits spread measurements of the same value among multiple servers, to load balance attributes that take on a small number of values, such as booleans. The middle and bottom bits are defined by an administrator-specified, per-attribute function that aims to balance load among keys based on expected values of that attribute. The only global configuration state on which all nodes must agree is the number of key bits used as attribute bits. Also, nodes issuing updates and queries must know the mapping function for each attribute they wish to update or query. A system with more or fewer attributes than are allowed by the number of configured attribute bits will function correctly, and any resulting sub-optimal load balance is corrected over long time scales using active load balancing in which DHT nodes reallocate the mapping of keys to responsible nodes [4].

We implement and evaluate four distributed range query approaches—namely **SingleQuery**, **MultiQuery**, **Index**, and **Fixed**—based on the preceding methodology. The first three use a DHT while the fourth does not. These techniques all return a list of nodes meeting a query's per-node requirements, and the corresponding measurements.

SingleQuery, illustrated in Figure 5(a), includes in each measurement report *all* attribute values for the reporting node. A reporting node transmits n measurement reports,

[1] We also implemented a scheme that follows DHT routing table pointers rather than successor pointers; we found that for our workloads, the two approaches performed within 10% of one another.

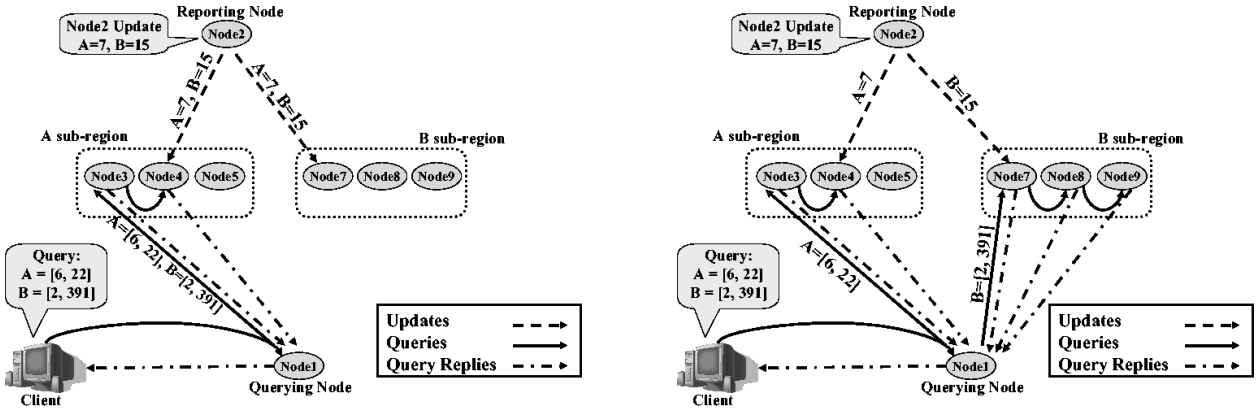

Figure 5. (a) The example query is routed to one key chosen randomly between the key for $A = 6$ and $A = 22$. It is then forwarded along DHT successor pointers to all other nodes responsible for values of A between 6 and 22. Those nodes return the lists of nodes they store matching both the A=[6,22] and B=[2,391] criteria. (b) The query is routed to one key between $A = 6$ and $A = 22$, and one between $B = 2$ and $B = 391$. It is then forwarded along DHT successor pointers as in (a). The nodes in the "A sub-region" return the lists of nodes they store matching the A=[6,22] criteria, and the nodes in the "B sub-region" return the lists of nodes they store matching the B=[2,391] criteria. The querying node intersects the node lists from the two sub-regions to find the set of nodes that match both criteria. In (a) and (b), all nodes except the client are part of SWORD and the DHT.

each containing n attribute values, to n servers, each of which is the DHT node responsible for a key derived from the value of one of the attributes as illustrated in Figure 4. While introducing relatively high overhead for updates, a multi-attribute query need only be sent to the set of servers responsible for the target range of one of the queried attributes. In particular, the query is first routed to a node responsible for any key in the query range of one attribute in the query (the "range search attribute," chosen with the goal of minimizing the number of DHT nodes that must be visited to satisfy the query), and it is forwarded along DHT successor pointers to all other nodes responsible for keys in the query range of that attribute. This technique is similar to Mercury [4] multi-attribute distributed range queries and eSearch [43] multi-keyword text search.

MultiQuery, shown in Figure 5(b), places a single value in each measurement report. A node reporting n attributes transmits n 1-attribute measurement reports to n DHT keys (servers). This approach uses less update bandwidth and storage than does SingleQuery. The downside is that one set of servers must be queried for each attribute in a query. Moreover, because each server only stores information about one attribute, each server can only filter on one attribute, so this approach can potentially return many more nodes than SingleQuery, in which each server can filter on all attributes. The query node sends a copy of the query to each server group in parallel and intersects the returned nodes to find those that satisfy all attributes in the query. This approach is inspired by the search strategy in XenoSearch [41] and keyword search in [38].

Fixed is an approximation to a non-replicated geographically distributed datacenter model with a varying number of servers. To send an update, a SWORD node sends one copy of its measurement report to one of n infrastructure servers that is assigned at random when the reporting node starts up (so that approximately m/n SWORD nodes are assigned to each server, when the reporting node population is m). A query node sends one copy of each query to that same server, which forwards it to the remaining $n-1$ servers, collects the results, and returns the results to the querying node. (Updating n nodes and querying one performed within 10% of this approach for our workloads.) A truly centralized solution is an instance of this approach with $n = 1$.

Index is a hybrid between the Fixed and SingleQuery approaches. Fixed infrastructure servers hold an index that maps contiguous DHT key ranges to the IP address of the DHT node responsible for that range. Updates are handled as with the SingleQuery approach. Queries are sent first to the index server(s) responsible for the key range of interest, and those index servers then forward the query directly to the DHT nodes responsible for the requested key range without first routing through the DHT. All queries are thus handled in three hops.

Our implementations use the Bamboo DHT [39], but the

approaches generalize to any DHT. In our DHT-based architectures we build our own soft-state distributed data repository on top of the key-based routing interface. Measurement reports time out after a fixed multiple of their update interval, thereby automatically deleting "dead" nodes and old measurements. Determining an appropriate measurement update rate depends on a number of factors, including measurement stability over time, required accuracy, and desired update bandwidth consumption. Experiments on PlanetLab showed that 94% of nodes meeting a typical load constraint (5-minute load average $<= 5$) at a given point in time continuously met it over the subsequent 5-minute interval, and that 89% of nodes meeting that load constraint at a given point in time continuously met it over the subsequent 10-minute interval, suggesting an update interval on the order of 5 minutes provides high accuracy for most nodes. Amount of per-node network transmit bandwidth showed similar behavior.

New attributes can be added to SWORD at runtime. When a new attribute is installed on a reporting node, SWORD itself can be used to distribute the attribute's identity and its <attribute,value>-to-key mapping function—nodes periodically query a special attribute that each node reports, whose associated value is the mapping function(s) that the reporting node has added to the system.

3.1.2. Storing and Retrieving Inter-node Attributes

The process we have described retrieves the identities of all reporting nodes matching all *per-node* requirements in the query, along with the values of those attributes. To resolve our queries we also need inter-node measurements. To reduce the resource consumption of gathering $O(N^2)$ inter-node measurements, SWORD leverages the observation that nodes typically fall into equivalence classes for inter-node attributes. For example, the latency between Node A in Autonomous System 1 (AS1) and Node B in AS2 is often approximately equal to the latency between any node in AS1 and any node in AS2. SWORD therefore allows arbitrary groups of nodes to define a "representative" node that collects inter-node measurements on their behalf. Choosing appropriate representatives is an orthogonal issue we do not address that might leverage existing work on network-aware clustering [6, 24].

Instead of storing inter-node measurements in the DHT, representatives store inter-node measurements themselves. This saves them the bandwidth of periodically publishing a potentially large number of inter-node measurements into the DHT. Upon receiving node reports R from the per-node attribute range query, the querying node issues a second distributed query to the representative nodes indicated in R to obtain the inter-node attribute(s) of interest among those representative nodes.

The mapping from a node to its representative is one of the attributes reported by each node. Therefore to bootstrap, each node need only know the identity of its own representative, much as nodes must typically know the IP address of their DNS server. When a representative node boots, it performs a standard SWORD distributed query to find the identities of all other representatives in the system, and begins measuring to them.

3.2. Optimizer Implementation

Once the per-node and inter-node measurements have been retrieved, the optimizer finds the lowest-penalty mapping of nodes to the groups specified in the query. The optimizer computation is not parallelized, running only on the node that received the initial query from the client. But because clients may contact any SWORD node initially, the computation is effectively distributed on a per-request basis. The task of creating groups of a specific size that satisfy inter-node and inter-group constraints is NP-hard. Our optimizer finds the penalty of all possible node combinations, but the optimizer biases this exponential search towards groups that are likely to have low penalty, so that the search can be terminated prior to completion without severely impacting result quality. This allows the user to trade off result quality for running time.

First the optimizer generates a set of "candidate groups" meeting the per-node and inter-node requirements for each group. To do this, for each group G in the query, the optimizer first enumerates all combinations of nodes meeting G's per-node requirements, and labels each combination with the total per-node penalty for that set of nodes. For each of these combinations, it then computes the total inter-node penalty (which is infinite if the set does not meet the requirements). Combinations of nodes that do not have an infinite penalty become candidate groups. SWORD allows the user to specify a time limit for this phase of the computation, and evaluates each combination in order of increasing labeled per-node group penalty. Thus if the time limit expires before the optimizer has evaluated all combinations, at the very least it has evaluated the sets of nodes with the lowest total per-node penalty; these sets generally also have the lowest total group penalty (once inter-node penalty is added).

Next the optimizer tests and ranks these candidate groups based on inter-group requirements and preferences. The algorithm is similar to that described above, but instead of searching for groups of nodes, it searches for groups of groups, starting with the lowest-penalty candidate groups.

The full exponential search for the lowest-penalty solution may take a long time to complete. Thus, we have implemented several simple heuristics for stopping the search early. **Three second timeout** stops the search after three

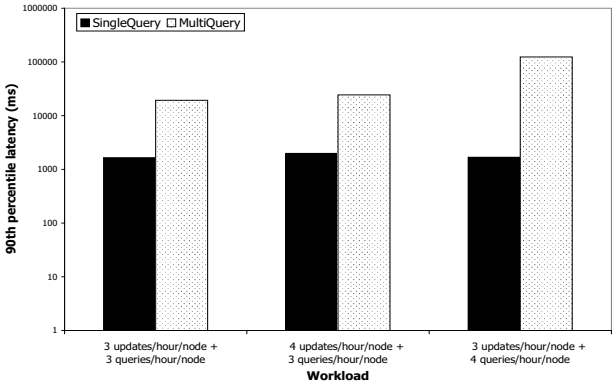

Figure 6. 90th percentile latency of SingleQuery vs. MultiQuery approaches, with workload of 3 or 4 updates/hour/node and 3 or 4 queries/hour/node.

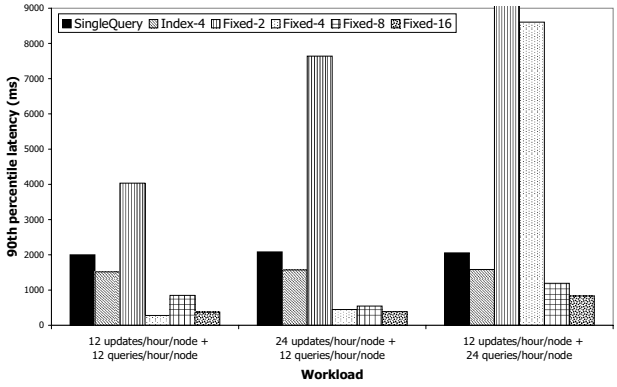

Figure 7. 90th percentile latency of SingleQuery, Fixed, and Index per-node attribute range query approaches. Workload is 12 or 24 updates/hour/node and 12 or 24 queries/hour/node.

seconds. In this case we divide the allotted time evenly between the first and second phases described above, and subdivide the first phase's time evenly among all groups G. **Top half of candidates** searches only the lowest-penalty half of the candidate groups. **Top 5 candidates** eliminates all but the lowest-penalty 5 candidates for each group before running the search. Finally, since the lowest penalty candidate groups are processed first in the search, the first answer found may be sufficient; thus the **first answer** heuristic stops once the first valid solution is found. We evaluate the performance and accuracy of these heuristics in Section 4.2.

4. Evaluation

4.1. Emulation-based Evaluation

Our emulation experiments focus on the performance and overhead of SWORD's distributed query processor. We choose latency as our performance metric because SWORD clients may periodically re-query SWORD to adapt their application to changing node and network conditions, or as the resource needs of their application changes. We choose aggregate bandwidth consumption as our overhead metric because it potentially represents a real financial cost to the users and operators of the infrastructure. This section evaluates our distributed query processor. We evaluate optimizer performance separately in Section 4.2, and end-to-end performance on PlanetLab in Section 4.3.

We evaluated SWORD's query processor on a cluster of 40 IBM xSeries PCs with Dual 1 GHz Pentium III processors, 1.5 GB of RAM, and Gigabit Ethernet. We used ModelNet [45] with an INET topology [5] with 10,000 transit nodes, 1,000 client nodes and 8 client nodes per stub. Transit-transit links were given 150 Mb/s (OC3) bandwidth and client-stub links 384 Kb/s bandwidth. Latencies were based on the INET topology. $1/32$ of the nodes were chosen as "representatives." When evaluating the Fixed and Index approaches, the infrastructure servers were grouped into 4-node stub domains each with 150 Mb/s, 1 ms latency connections to their upstream transit node, to emulate an environment in which a service provider has distributed servers among a number of geographically distributed, well-connected co-location centers. For the DHT-based approaches, we used the Bamboo DHT available in August 2004.

Our baseline workload consisted of updates (measurement reports) and queries issued by each of 1000 virtual nodes. The content of updates were taken from a representative one-hour portion of a trace of Ganglia [40] and Vivaldi [11] measurements collected from PlanetLab. Updates contained 32 metrics collected by Ganglia during that time period, along with network coordinates computed by Vivaldi and several attributes we used for debugging, for a total of 40 attributes. Queries contained five per-node attributes—fifteen-minute load average, free disk space, free memory, bytes per second received, and bytes per second sent—and one inter-node attribute—inter-node latency.

Queries were formulated according to a distribution such that they requested a minimum amount of disk space that is Zipf distributed between 10 MB and 100 MB (biased toward the high end of the range), a fifteen-minute load average less than a uniformly distributed value between 0 and 5.0, a minimum amount of free memory that is Zipf distributed between 0 MB and 48 MB (biased toward the high end of the range), bytes per second in and out (competing traffic) that is no more than 0.1 MB/s for half of the queries and unconstrained for the other queries, and inter-node latency

Range Search Technique	Normalized bandwidth
Fixed-2	1.0
Fixed-4	1.22
Fixed-8	1.37
Fixed-16	1.34
Index-4	2.10
SingleQuery	2.22

Table 1. Bandwidth consumption for the 12/12 workload, normalized to Fixed-2, which consumed an average of 5 Mb/s during the one-hour run. Network traffic overhead includes measurement reports, queries for per-node attributes, retrieving inter-node attributes, maintaining the DHT, and periodically computing network coordinates.

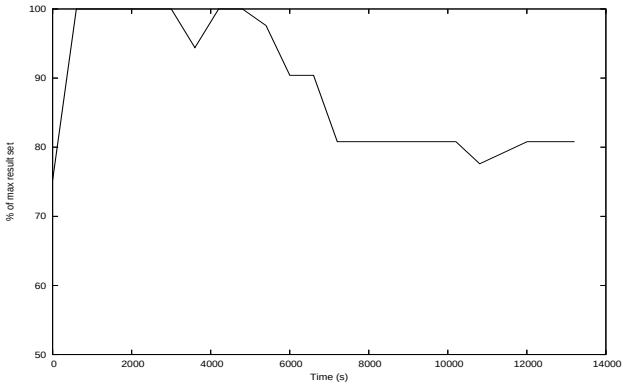

Figure 8. Number of candidate nodes returned by range query as a function of time, before and after killing 20% of the reporting nodes at 5000 seconds.

between 0 ms and 1000 ms. Because our trace contained valid data for only 124 PlanetLab nodes, we emulated a 1000-node system by duplicating each of the 124 entries an average of 8 times. The median number of nodes returned per query was 120 (12% of the nodes in the total system) and the 90th percentile was 160. In the DHT approaches, attribute bits are assigned so that each of the 40 attributes is mapped to a sub-region containing 25 nodes.

4.1.1. Distributed Query Latency

Figure 6 shows the impact of range-search approach and workload intensity on the latency to satisfy the range query for SingleQuery compared to MultiQuery. At higher workload rates, our emulation cluster's CPUs became overloaded for the MultiQuery approach. Nonetheless, these experiments reveal that SingleQuery clearly outperforms MultiQuery for our "typical" resource discovery workload, so we do not consider MultiQuery further. The primary reason for the difference is that the network bandwidth consumed by the larger number of nodes returned to the querying node by MultiQuery compared to SingleQuery creates sufficient congestion to overwhelm the benefit MultiQuery derives from sending only one attribute per update.

Figure 7 shows the impact of range-search approach and workload intensity on the latency to satisfy the range query for SingleQuery compared to the remaining approaches. We see that Index always outperforms the other DHT-based approaches. This is reasonable because queries in Index take three hops in parallel (one to the index server(s), one to the DHT server(s) storing measurements, and one back to the querying node) while in the SingleQuery approaches each query may visit up to 25 nodes (the maximum number of nodes to which any attribute's range is mapped). The Fixed approaches vary greatly in performance, but we see that for our workload, a fixed infrastructure cluster with a relatively small number of nodes and a high-bandwidth network connection can better support the "typical" resource discovery workload that we tested than can an infrastructure based on end-nodes organized into a DHT. The Fixed configurations that performed poorly did so because of congestion.

In another set of experiments, we varied the number of representatives. We found that electing half of the nodes as representatives reduces query latency by up to 70% for the Fixed approaches and 7% for the SingleQuery approach compared to electing all nodes as representatives; the improvement increases to 90% and 10% when a quarter of nodes are representatives.

4.1.2. Bandwidth Consumption

Table 1 shows SWORD's bandwidth consumption. The centralized (Fixed) approaches use the least bandwidth. This is because they send one copy of each 40-attribute measurement report to one server rather than one copy to each of 40 servers as in the Index and SingleQuery approaches. Also, the query in Fixed must on average be distributed to a smaller number of nodes (2, 4, 8, and 16), rather than up to 25 for the DHT-based approaches. Electing half of the nodes as representatives reduces bandwidth consumption by up to 15% compared to electing all, and electing one quarter reduces it by up to 23%.

4.1.3. Robustness to Perturbations

SWORD takes advantage of the DHT's self-healing property to automatically remap keys to nodes when a node fails or recovers, or voluntarily joins or leaves the system. To verify this robustness mechanism, we applied a workload

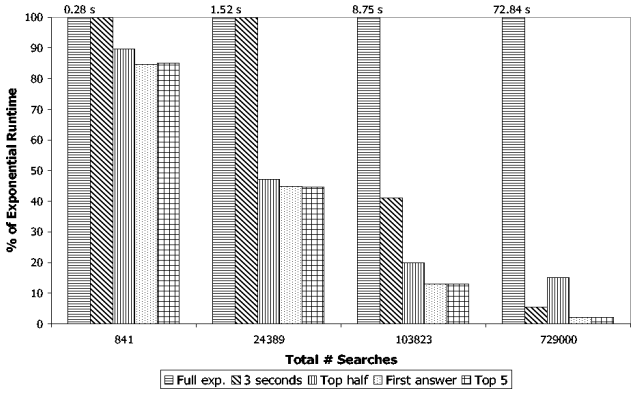

Figure 9. Runtime of optimizer using different heuristics shown as a percentage of the runtime to complete the full exponential search. The x-axis shows the total number of possible group combinations that would be checked if the complete exponential search were run. Total runtime for the exponential search is indicated at the top of the bars.

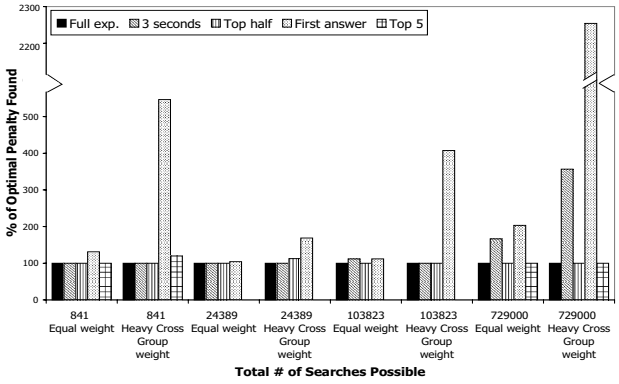

Figure 10. Accuracy of optimizer heuristics relative to optimal solution found in full exponential search. x-axis shows total number of group combinations checked if the full exponential search were run. A missing bar indicates that no solution was found.

to SWORD (in the SingleQuery configuration), and killed 20% of the DHT nodes 5000 seconds into the run.

Figure 8 plots percentage of the maximum result set returned during each 10 minute interval. All nodes are "servers" in the DHT storing measurement reports as well as load generators, so killing 20% of the nodes removes 20% of the reporting nodes, and we therefore expect 80% of the initial (maximum) result set to be returned once the system recovers. Indeed we see soon after 20% of the nodes are killed at time 5000 seconds, queries begin receiving the new result set (containing 80% of the original result set) once Bamboo "heals," stale data times out of SWORD, and new measurement reports are issued.

4.2. Optimizer Performance

We measured the performance of the optimizer on a single 3 GHz Pentium 4 node. The update workload came from Ganglia and all-pairs-pings [42] measurements on PlanetLab, and the query workload consisted of a representative mix of queries containing 2 or 3 groups.

Figure 9 shows the running time of the optimizer using each of our heuristics, as a percentage of the optimizer running time when using the full exponential search. We see that for larger problems, the savings gained from using a heuristic is significant, reducing a 72 second search to a few seconds or less.

Reducing the running time of the search is useful only if the result returned maintains an acceptable level of accuracy. Figure 10 shows the accuracy of the various heuristics relative to the results of the full exponential search. In Figure 10, the "equal weight" bars represent queries in which penalties were assigned to all attributes equally. The "heavy cross-group weight" bars represent queries in which the penalty assigned to the cross-group constraint is 10 times greater than the other attributes. We see that the 3-second heuristic performs well for small searches, that the "top half" heuristic performs well in all cases and, for our workload, actually finds the optimal solution. The "first answer" approach is the least accurate heuristic, and the "top 5" heuristic does not find a feasible solution at all in half of our test cases.

4.3. End-to-end Performance on PlanetLab

Compared to our ModelNet configuration, PlanetLab has a smaller number of nodes and more CPU contention. We ran experiments on PlanetLab on July 16, 2004 on the following two sets of nodes (one a subset of the other): i) all 214 usable nodes that were connected to the commodity Internet, and ii) a subset of the first set that are all at universities in North America and tend to have high-bandwidth, low-latency network paths to one another. We used SWORD in the SingleQuery configuration. Nodes report the same metrics as in our ModelNet experiments. We measured query latency for a single query at a time; so the measured times show "best case" latency.

We issued a series of queries, each requesting two groups of 4 nodes each, such that the inter-node latency among all nodes within each group was between 0 ms and 150 ms, and the load on each node was between 0 and N, where N was varied from query to query to cover all integers between 1

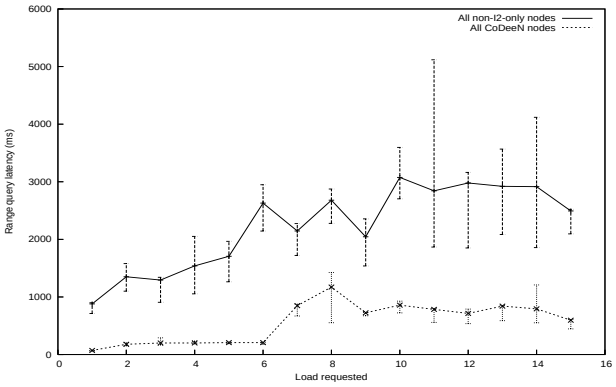

Figure 11. Range query median latency versus upper bound on requested load, on PlanetLab. Number of candidate nodes returned ranged from 108 to 214 for the top line and 34 to 108 for the bottom line, with increasing numbers of nodes returned as the upper bound on the requested load was increased.

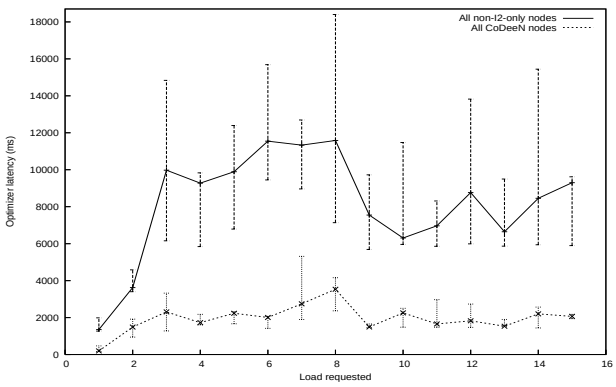

Figure 12. Optimizer median latency versus upper bound on requested load, on PlanetLab. This data is from the same experiment as in Figure 11.

and 15, inclusive.

Figure 11 shows that SWORD's range search performs reasonably well on PlanetLab, returning results to the optimizer within a few seconds even when all nodes are returned by the range query. Figure 12 shows that optimizer running time for the full exponential search ranged from two to twelve seconds. The completion time appeared to be more strongly correlated with the load on the machine performing the optimization than on the size of the optimization problem.

SWORD is currently used for resource discovery by two other PlanetLab services, Bellagio [1] and PLuSH [44].

In sum, end-to-end, even without shortcutting the distributed query or optimization steps to sacrifice accuracy for latency, we find that typical queries can be resolved in less than five seconds in an emulated 1000-node system, and in less than ten seconds on PlanetLab, with the larger PlanetLab latency caused by CPU load on the node performing the optimization step.

5. Related Work

SWORD builds on work in resource discovery, Internet-scale query processing, and distributed range search.

Kee *et al.* describe "virtual grids" [23]. The description language vgDL allows users to describe resource requirements as hierarchies of homogeneous or heterogeneous groups of nodes with good or poor connectivity, reminiscent of SWORD's groups with per-node, inter-node, and inter-group constraints, but with coarser-grained specifications and support for arbitrarily deep hierarchies. The resource mapping component vgFAB stores resource measurements in a centralized database in contrast to SWORD's distributed architecture, and it computes a bounded set of pre-fabricated groups and stores them in the database rather than dynamically forming them on a query-by-query basis.

Condor and its ClassAds language [27] provide similar functionality to virtual grids and SWORD, absent the notion of groups and inter-node connectivity constraints. Gang matching [37] extends Condor's original bilateral matching scheme to a multilateral one, allowing co-allocation of multiple resources. Set matching [29] allows requests that express aggregate constraints. Redline [28] formulates the matching problem as a constraint satisfaction problem. These latter systems allow expression of resource groups, but they do not offer a concise method to express network topologies. Also, to date their implementations have been centralized. R-GMA [15] and RGIS [12] use a relational model to track and query dynamic and static per-node attributes, respectively. The latter features nondeterministic queries, allowing users to trade reduced response time for number of results received.

XenoSearch [41] supports DHT-based multi-attribute range queries in a manner similar to our MultiQuery approach, but it uses a separate DHT instance per attribute, creates its query routing structure explicitly rather than using built-in DHT successor pointers, and provides approximate answers using Bloom Filters. Additionally, SWORD allows users to define groups with inter-node and inter-group requirements and "penalty functions" to rank nodes meeting the requirements.

Globus MDS2 [49] allows GIIS indexing servers to aggregate measurement data about GRIP information providers obtained from GRIS. SWORD's query processor could be used as a GIIS, connecting GRISes in a peer-to-

peer fashion. MDS3 and MDS4 have recently emerged as successors to MDS2.

PIER [20], Sophia [47], IrisNet [30], and Astrolabe [46] provide Internet-scale query processing. All four could be used to satisfy per-node resource queries, and they offer a more expressive language for specifying such requirements than SWORD. However, the first three must contact all data-storing nodes to perform range search and the last disseminates measurement data globally, while SWORD targets its range search to only the nodes storing measurements within the target attribute's range.

DHT-based range search was suggested initially by Karger and Ruhl [21], and was later implemented and enhanced in Mercury [4]. Our SingleQuery approach is similar to Mercury, but with additional "passive" load balancing provided by the <attribute,value>-to-DHT-key functions. PHT [35] offers an alternative range search strategy based on tries on top of DHTs.

The network topology embedding problem is formulated as a constraint satisfaction problem in [8] for wide-area networks and as an optimization problem in [48] for cluster networks.

6. Conclusion

We have described SWORD, a scalable resource discovery service for wide-area distributed systems. Users define a requested system topology in terms of groups with required intra-group, inter-group, and per-node characteristics whose relative importance and sensitivity are expressed using penalty functions. We explore a number of distributed query algorithms for finding nodes meeting required per-node constraints, and several heuristics for finding the best mapping of nodes to groups. Our evaluation shows that a fixed server cluster at network peering facilities typically outperforms a DHT-based resource discovery infrastructure. Nonetheless, we find that a fully decentralized version of SWORD in emulation and on PlanetLab performs reasonably well, while benefiting from the DHT's resilience. While our results are specific to the architectures and workloads we examined, we believe that our experience considering a variety of architectures provide interesting insights regarding appropriate architectures for a variety of systems depending on available resources, expected level of load, and required levels of performance and availability.

An important area of future work is security. Nodes could sign measurement reports and queries as a form of authentication. Given an authentication infrastructure, per-node rate limiting could ensure that no node utilizes more than a predefined amount of bandwidth (or optimizer CPU time) per unit time on any single node. We note, however, that such a technique is vulnerable to the Sybil attack [13] and therefore requires a trusted identity creation service. To ensure that nodes are truthful in their measurement reports, a verification service could run micro-benchmarks to verify that resource availability matches earlier advertisements. To ensure that, modulo collusion, nodes are truthful when they run the optimizer, a client might issue each query to several query nodes and compare the results.

Privacy is the most challenging security issue for distributed versions of SWORD. Reporting nodes could encrypt attribute names to hide their values, but our range search mechanism relies on a monotonic mapping function from measured values to DHT keys, and encrypting values using standard techniques, either before or after mapping them to a DHT key, will break this monotonicity. Privacy-preserving DHT-based range search is an interesting topic for future work.

Finally, we have not yet studied the system dynamics that result from multiple large-scale applications periodically querying SWORD to determine when and how to migrate application instances. We anticipate that mechanisms are needed to damp potential oscillations.

SWORD's PlanetLab deployment can be accessed at http://www.swordrd.org/.

References

[1] A. AuYoung, B. N. Chun, A. C. Snoeren, and A. Vahdat. Resource Allocation in Federated Distributed Computing Infrastructures. In *OASIS '04*, October 2004.

[2] M. Balazinska, H. Balakrishnan, and D. Karger. INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery. In *International Conference on Pervasive Computing*, August 2002.

[3] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating Systems Support for Planetary-Scale Network Services. In *NSDI*, 2004.

[4] A. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting Scalable Multi-Attribute Range Queries. In *Proceedings of SIGCOMM 2004*, 2004.

[5] H. Chang, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. Towards Capturing Representative AS-Level Internet Topologies. In *SIGMETRICS*, June 2002.

[6] Y. Chen, D. Bindel, H. Song, and R. H. Katz. An algebraic approach to practical and scalable overlay network monitoring. In *ACM SIGCOMM 2004*, 2004.

[7] W. Chrabakh and R. Wolski. Gridsat: A chaff-based distributed sat solver for the grid. In *Supercomputing*, 2003.

[8] J. Considine, J. Byers, and K. Mayer-Patel. A Constraint Satisfication Approach to Testbed Embedding Services. In *Proceedings of ACM HotNets-II*, November 2003.

[9] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proceedings of HPDC '01*, 2001.

[10] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke. SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management

in Distributed Systems. In *Lecture Notes in Computer Science*, number 2537, pages 153–183, 2002.

[11] F. Dabek, R. Cox, F. Kaahoek, and R. Morris. Vivaldi: A Decentralized Network Coordinate System. In *SIGCOMM '04*, 2004.

[12] P. Dinda and D. Lu. Nondeterministic queries in a relational grid information service. In *Supercomputing '03*, 2003.

[13] J. R. Douceur. The sybil attack. In *IPTPS*, 2002.

[14] Eman introduction. http://ncmi.bcm.tmc.edu/homes/stevel/EMAN/EMAN/doc/intro.html, 2004.

[15] S. Fisher. Relational model for information and monitoring. Technical Report GWD-GP-7-1, Grid Forum, 2001.

[16] M. J. Freedman, E. Freudenthal, and D. Mazihres. Democratizing content publication with coral. In *NSDI*, 2004.

[17] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat. SHARP: An Architecture for Secure Resource Peering. In *SOSP*, October 2003.

[18] R. Gardner et al. The grid2003 production grid: Principles and practice. In *HPDC*, 2004.

[19] A.-C. Huang and P. Steenkiste. Network-sensitive service discovery. In *USITS*, 2003.

[20] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *Proceedings of VLDB*, September 2003.

[21] D. Karger and M. Ruhl. Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems. In *IPTPS*, 2004.

[22] B. Karp, S. Ratnasamy, S. Rhea, and S. Shenker. Spurring Adoption of DHTs with OpenHash, a Public DHT Service. In *IPTPS*, February 2004.

[23] Y.-S. Kee, D. Logothetis, R. Huang, H. Casanova, and A. Chien. Efficient resource description and high quality selection for virtual grids. Technical Report CS2004-0809, University of California, San Diego, 2004.

[24] B. Krishnamurthy and J. Wang. On Network-Aware Clustering of Web Clients. In *SIGCOMM*, 2000.

[25] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An Architecture for Global-scale Persistent Storage. In *Proceedings of ACM ASPLOS*, November 2000.

[26] W. Li, R. Byrnes, J. Hayes, A. Birnbaum, V. Reyes, A. Shahab, C. Mosley, D. Pekurovsky, G. Quinn, I. Shindyalov, H. Casanova, L. Ang, F. Berman, P. Arzberger, M. Miller, and P. Bourne. The encyclopedia of life project. *New Generation Computing*, 22(2), February 2004.

[27] M. Litzkow, M. Livny, and M. Mutka. Condor - A Hunter of Idle Workstations. In *ICDCS*, 1988.

[28] C. Liu and I. Foster. A constraint language approach to matchmaking. In *International workshop on Research Issues on Data Engineering (RIDE 2004)*, 2004.

[29] C. Liu, L. Yang, I. Foster, and D. Angulo. Design and evaluation of a resource selection framework. In *HPDC-11*, 2002.

[30] S. Nath, Y. Ke, P. B. Gibbons, B. Karp, and S. Seshan. IrisNet: An Architecture for Enabling Sensor-Enriched Internet Services. Technical Report IRP-TR-03-04, Intel Research Pittsburgh, June 2003.

[31] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Scalable wide-area resource discovery. Technical Report UCB//CSD-04-1334, UC Berkeley, 2004.

[32] Overview of mead. http://www.ncsa.uiuc.edu/expeditions/MEAD/MEADoverview.pdf, 2004.

[33] V. Pai. CoTop: A Slice-Based Top for PlanetLab. http://codeen.cs.princeton.edu/cotop/.

[34] V. S. Pai, L. Wang, K. Park, R. Pang, and L. Peterson. The dark side of the web: An open proxy's view. In *Proceedings of HotNets-II*, 2003.

[35] S. Ramabhadran, S. Ratnasamy, J. M. Hellerstein, and S. Shenker. Prefix hash tree. In *PODC*, 2004.

[36] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *HPDC*, 1998.

[37] R. Raman, M. Livny, and M. Solomon. Policy driven heterogeneous resource co-allocation with gangmatching. In *HPDC-12*, 2003.

[38] P. Reynolds and A. Vahdat. Efficient Peer-to-Peer Keyword Searching. In *Proceedings of the International Middleware Conference*, June 2003.

[39] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a dht. In *Proceedings of the USENIX Annual Technical Conference*, 2004.

[40] F. D. Sacerdoti, M. J. Katz, M. L. Massie, and D. E. Culler. Wide Area Cluster Monitoring with Ganglia. In *Proceedings of the IEEE Cluster 2003 Conference*, 2003.

[41] D. Spence and T. Harris. Xenosearch: Distributed resource discovery in the xenoserver open platform. In *Proceedings of HPDC*, 2003.

[42] J. Stribling. Planetlab all pairs pings. http://www.pdos.lcs.mit.edu/ strib/pl_app/, 2004.

[43] C. Tang and S. Dwarkadas. Hybrid Global-Local Indexing for Efficient Peer-to-Peer Information Retrieval. In *NSDI*, 2004.

[44] C. Tuttle, A. C. Snoeren, and A. Vahdat. Plush: A tool for remote deployment, management, and debugging. Submission to WORLDS 2004.

[45] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker. Scalability and Accuracy in a Large-scale Network Emulator. Technical report, Duke University, May 2002.

[46] R. van Renesse, K. Birman, and W. Vogels. Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining. *ACM TOCS*, 21(2):164–206, 2003.

[47] M. Wawrzoniak, L. Peterson, and T. Roscoe. Sophia: An Information Plane for Networked Systems. In *Proceedings of ACM HotNets-II*, November 2003.

[48] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *OSDI*, 2002.

[49] X. Zhang and J. Schopf. Performance analysis of the globus toolkit monitoring and discovery service, mds2. In *Proceedings of the International Workshop on Middleware Performance (MP 2004)*, April 2004.